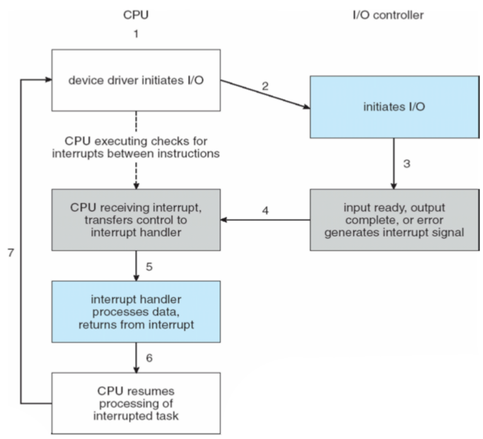
I/O Systems

Device Management — Objectives

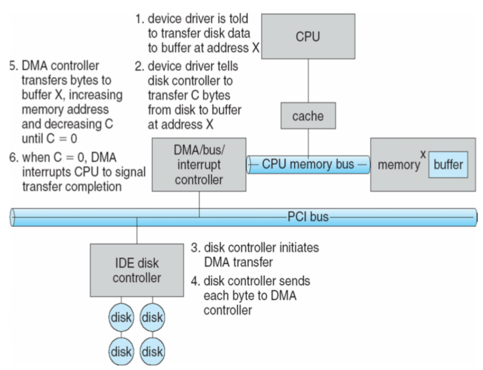
- **Abstraction** from details of physical devices
- **Uniform naming** that does not depend on hardware details
- **Serialization** of I/O operations by concurrent applications
- **Protection** of standard-devices against unauthorized accesses
- **Buffering** if data from/to device cannot be stored in final destination
- **Error handling** of sporadic device errors
- **Virtualizing** physical devices via memory + time multiplexing

Device Management — Techniques

- **Programmed I/O**:
  - thread is busy-waiting for I/O operation to complete → CPU cannot be used elsewhere
  - kernel is *polling* state of I/O device (command-ready, busy, error)
- **Interrupt-driven I/O**:
  - I/O command is issued
  - processor continues executing instructions
  - I/O device sends interrupt when command is done



- **Direct Memory Access** (DMA):
  - DMA module controls exchange of data between main memory and I/O device
  - processor interrupted after entire block has been transferred
  - → bypasses CPU to transfer data directly between I/O device and memory



Kernel I/O Subsystem

- **Scheduling**: order I/O requests in per-device queues
- **Buffering**: store data in memory while transferring between devices
- **Error handling**: recover from read/availability/write errors
- **Protection**: protect from accidental/purposeful disruptions
- **Spooling**: hold output to device if device is slow (e.g., printer)
- **Reservation**: provide exclusive access for process

Device Drivers

- **Jobs**:
  - *translate* user request through device-independent standard interface
  - *initialize* hardware at boot time
  - *shut down* hardware

Device Buffering

- **Reasons**:
  - without buffering threads must wait for I/O to complete before proceeding
  - pages must remain in main memory during physical I/O
- **Version 1 — block-oriented**:
  - information is stored in fixed-size blocks
  - transfers are made a block at a time
  - used for disks/tapes
- **Version 2 — stream-oriented**:
  - transfer information as byte stream
  - used for keyboard, terminals, …(most things that is not secondary storage)

Buffering — User level

- **Principle**: task specifies memory buffer where incoming data is placed
- **Issues**:
  - what happens if buffer is currently paged out to disk? → data loss
  - additional problems with writing? → when is buffer available for re-use?

Buffering — Single

- **Principle**: user process can process one data block while next block is read in
- **Swapping**: can occur since input is taking place in system memory, not user memory
- **Stream-oriented**: buffer = input line, carriage return signals end of line
- **Block-oriented**:
  - input transfers made to *system buffer*
  - buffer moved to *user space* when needed
  - another block read into system buffer

Buffering — Double

- **Principle**: use 2 system buffers instead of 1 (per user process)
- user process can write/read from one buffer while OS empties/fills other buffer

Buffering — Circular

- **Problem**: double buffer insufficient for high-burst traffic situations:
  - many writes between long periods of computations
  - long computation periods while receiving data
  - might want to read ahead more than just single block from disk